Ali Malik
CS106L Winter 2018

Handout 03
Feb 1st, 2017

Assignment 2: CS106L WikiRacer

Due: Saturday, 10th Feb, 11:59pm

Introduction:

Human beings are obsessed with finding patterns. Whether it be in the depths of mathematical study or the playground of hobbies like Chess and Sudoku, it is clear that the human race enjoys engaging with the art and science of discovering trends. It is actually claimed by some that our ability to recognize the most complex of patterns was one of the central factors in our development as an ultra intelligent species. Yet with the recent advances in data mining and information distribution, we have reached an age where the sheer volume of data easily overwhelms our ability to immediately understand it. This is where you, as computer programmers, come in. Just as the advances in computing have allowed us to gather such egregiously large collections of data, so to have advances in algorithm design and programming methodology pushed the boundaries of the complexity of patterns we can detect. With the advent of fields like data science, we are now able to inspect and analyze trends beyond those capable by our minds alone.

One interesting place to look for meaningful trends is the vast collection of articles on Wikipedia. For example, there is a famous observation that repeatedly clicking the first, non-parenthesised/italicised link starting on any Wikipedia page will always get you to the Philosophy page. This fact, popularised by an xkcd comic's hover caption, lead a team of mathematicians from the University of Vermont to conjecture that the flow of information in the world's largest, most meticulously indexed collection of human knowledge tends to pages like Philosophy because the subject matter of this field is a major organizing principle for the ideas represented by human knowledge. Trends like this, where a collection of information conceives a web of emerging relationships, can give enormous insight into the structure of human knowledge and understanding.

One fun game to play is Wikiraces, where any number of participants race to get to a target Wikipedia page by using links to travel from page to page. The start and end page can be anything but are usually unrelated to make the game harder. Before the timer starts, you are allowed some time to read the target page to get a better understanding of it. If you want to have a try, there is an online version here!

Although usually just a fun past time, looking at different Wikipedia ladders can give a lot of interesting insights into the relationship and semantic similarity between different pages. In this assignment, you are going to implement a bot that will intelligently find a Wikipedia link ladder between two given pages. In the process you will get practice working with iterators, algorithms, templates, and special containers like a priority queue.

A broad pseudocode overview of our algorithm is as follows:

```
To find a ladder from startPage to endPage:
     Make startPage the currentPage being processed.

     Get set of links on currentPage.

     If endPage is one of the links on currentPage:
          We are done! Return path of links followed to get here.

     Otherwise visit each link on currentPage in an intelligent way
     and search each of those pages in a similar manner.
```

To simplify the implementation and allow for some testing, we will do this in two parts. The greyed out last step in the pseudocode will be the task of Part B.

Preliminary Task:

This is a really quick part that just involves taking a few screenshots and emailing them to me. This will let me test how your computer is dealing with internet connectivity, which you can imagine is going to be important!

The starter code contains two projects: InternetTest and WikiRacerLinks. For this preliminary task, just open the InternetTest project in QT and build and run it. This should prompt you with a console with a bunch of text. Every time the console asks you to "take screenshot and press enter to continue", take a screenshot, and press enter! There should be 4 screenshots in total. If you get any compiler errors, or anything strange, please screenshot those too. At the end, you can just email all the screenshots to me, *with a mention of your OS and Qt version*, and that's it :)

Please do get this part of the assignment to me immediately - if any issues come up, I will need some time to patch them up.

Make the subject line of the email:

Assignment2_screenshots

Part A:

Note: *This part should be done in the main.cpp file in the WikiRacerLinks project. Don't write this in the InternetTest project - that part is only for the screenshots.*

As you can imagine, a really important part of our code for this assignment will involve taking a Wikipedia page's html and returning a set of all links on this page. Part A of the assignment will be to implement a function

```
unordered_set<string> findWikiLinks(const string& page_html);
```

that takes the html of a page in the form of a string as a parameter and returns an unordered_set<string> containing all the ***valid Wikipedia links*** in the page_html string. For those who don't remember, an unordered_set behaves exactly like a set but is faster when checking for membership (in the Stanford library it is called a HashSet).

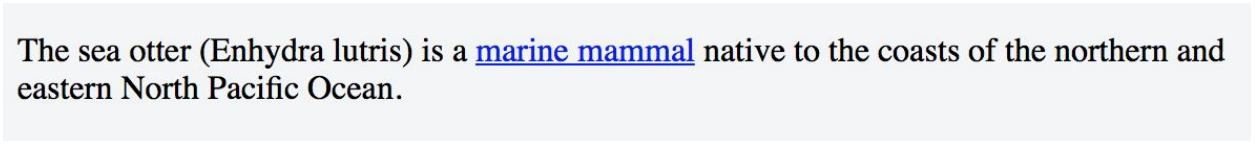
HTML:

First a quick overview of HTML. HTML (Hypertext Markup Language) is a language that lets you describe the content and structure of a web page. When a browser loads a webpage, it is interpreting the content and structure described in the HTML of the page and displaying things accordingly.

We won't go into exceptional detail over how HTML works but we will discuss how hyperlinks are formatted. Here is how a section of HTML on a Wikipedia page might look:

```
<p>
The sea otter (Enhydra lutris) is a
<a href="/wiki/Marine_mammal">marine mammal</a> native to the
coasts of the northern and eastern North Pacific Ocean.
</p>
```

which would display the following:



The sea otter (Enhydra lutris) is a marine mammal native to the coasts of the northern and eastern North Pacific Ocean.

Here we can see that to specify a piece of text as hyperlink, we have to surround it with the <a href="target"></a> tag. This would look like this:

```
<a href="link_path">link text</a>.
```

Valid Links:

For our intents and purposes, a valid Wikipedia link is any link satisfying these two properties:

1. The link is of the form "/wiki/PAGENAME".
2. The PAGENAME doesn't contain any of the disallowed characters (# or :).

The aim of this method will be to find all valid Wikipedia links of this form in the input html string and return an unorderd_set containing just the PAGENAME part of the link.

A quick example might clarify this:

```
<p>
In <a href="/wiki/Topology">topology</a>, the <b>long line</b> (or
<b>Alexandroff line</b>) is a
<a href="/wiki/Topological_space">topological space</a> somewhat
similar to the <a href="/wiki/Real_line">real line</a>, but in a
certain way "longer". It behaves locally just like the real line, but
has different large-scale properties (e.g., it is neither
<a href="/wiki/Lindel%C3%B6f_space">Lindelöf</a> nor
<a href="/wiki/Separable_space">separable</a>). Therefore, it serves
as one of the basic counterexamples of topology
<a href="http://www.ams.org/mathscinet-getitem?mr=507446">[1]</a>.
Intuitively, the usual real-number line consists of a countable
number of line segments [0,1) laid end-to-end, whereas the long line
is constructed from an uncountable number of such segments. You can
consult
<a href="/wiki/Special:BookSources/978-1-55608-010-4">this</a> book
for more information.
</p>
```

In this case, our method would return an unordered_set containing the following strings

```
    Topology
    Topological_space
    Real_line
    Lindel%C3%B6f_space
    Separable_space
```

Note two things of interest here. Firstly, the method would not return the links highlighted as red because they are not valid Wikipedia links. In the first case it is not of the form /wiki/PAGENAME and in the second case, the link contains the invalid character `':'`. Secondly, note that the `Lindelöf` link seems to have weird percentage signs and letters in it's hyperlink. This is how the html handles non-standard ascii characters like `'ö'` so you should just leave it as it is. Don't worry about cleaning it up!

Additional Rules:

For this assignment, I'm going to mandate that you **don't** use indices or the builtin string methods like **string.find** to do the searching through the text. Instead, look at algorithms like std::search and std::find that operate on iterators.

The reason for this is twofold. Firstly, I want you to get practice using iterators and algorithms and since the string methods deal with indices, you will be missing out on this learning opportunity. Secondly, arguments are made that the string methods exist only for legacy reasons. Few other containers provide their own specialised find function and string only does this because it existed before the STL was worked into the C++ standard library.

Implementation Tips:

- The code we wrote on the February 6th lecture is really helpful for this part of the assignment. We wrote a method that sequentially looked through a text for instances of a string. That same approach is applicable to solving this problem.

- There are some test files in the res folder that you can find the links of and compare with the expected output files. Start with the smaller files like simple.txt and simple-invalid.txt before moving up to the bigger ones!

- My solution uses the following algorithms:

  - std::search // find the start of a link
  - std::find // find the end of a link
  - std::all_of // see if link contains invalid characters

  I would definitely recommend you try your best to see where you can leverage these. I have commented some potential use cases.

If you want to discuss your plan of attack or discuss where an algorithm might be useful, definitely email me! I look forward to seeing what you come up with :)

Part B: To be released later